BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

EXHIBIT B

System.Firewall.P licy.P licyC nditi n

```
namespace System.Firewall.Policy
{
    public abstract class PolicyCondition : PolicyObject
    {
    public abstract bool Equals(PolicyCondition condition);
    public abstract bool Intersects(PolicyCondition condition);
    public abstract bool Contains(PolicyCondition condition);
    }
}
```

Method	
Name	Equals
Return Type	Bool
Description	Return true if all packets that match this PolicyCondition object also match the passing argument *condition* and vice versa. Otherwise return false.
Parameters	PolicyCondition cond

Method	
Name	Intersects
Return Type	Bool

Descripti n	Return true if there are packets that match this PolicyCondition object also match the passing argument *condition*. Return false if there is no such packet.
Parameters	PolicyCondition

Method	
Name	Contains
Return Type	Bool
Description	Return true if all packets that match the passing argument *condition* also match this PolicyCondition object. Otherwise return false.
Parameters	PolicyCondition

System.Firewall.Policy.EthernetConditi n

```
namespace System.Firewall.Policy
{
    public abstract class LinklayerCondition : PolicyCondition
    {
        public enum LinkLayer
        {
            InboundTop,
            OutboundTop,
            InboundBottom,
            OutboundBottom
        }
        // Properties
        public LinkLayer Layer { get { } set { } }
    }
        public class EthernetCondition : PolicyCondition
        {
        public MACAddressValue SourceMACAddress { get { } set { } }
        public MACAddressValue DestinationMACAddress { get { } set { } }
        public EthernetCondition();
        public EhternetCondition(LinkLayer layer, MACAddressValue src, dst);
        public override bool Equals(PolicyCondition val);
        public override bool Intersects(PolicyCondition val);
        public override bool Contains(PolicyCondition val);
        }
}
```

Property	
Name	Layer
Description	Specify where this link layer rule will be applied: • **InboundTop**: This layer is called after each received packet has traversed all other NDIS light weight filter shims. On the receive path, this is the last chance to filter a data-link packet before it is delivered to the network layer for processing. • **OutboundTop**: This layer is called before each sent packet has traversed any other NDIS light weight filter shims. On the send path, this is the first chance to filter a data-link packet before it is processed by other NDIS light weight filters. • **InboundBottom**: This layer is called before each received packet has traversed any other NDIS light weight

	filter shims. This layer is the first opportunity to filter a received packet. • **OutboundBottom**: This layer is called after each sent packet has traversed all other NDIS light weight filter shims. This layer is the last opportunity to filter a sent packet.
Access	Read/Write

Property	
Name	SourceMACAddress
Description	This value is used to match the source MAC address field in the Ethernet header.
Access	Read/Write

Property	
Name	DestinationMACAddress
Description	This value is used to match the destination MAC address field in the Ethernet

	header.
Access	Read/Write

System.Firewall.P licy.IPC ndition

```
namespace System.Firewall.Policy
{
	public class IPCondition : PolicyCondition
	{
		public enum IPLayer {
			InboundIPPacket,
			OutboundIPPacket,
			InboundIPFragment,
			OutboundIPFragment,
			IPForward
			};
			public IPLayer Layer { get { } set { } }
			public IPAddressValue SourceAddress { get { } set { } }
			public IPAddressValue DestinationAddress { get { } set { } }
			public ByteValue Protocol { get { } set { } }
			public UInt16Value PacketLength { get { } set { } }
		public NetworkInterface Interface { get { } set { } }
			// There may be more conditions to be exposed by the firewall
platform.
		public IPCondition();
		public IPCondition(IPLayer layer, IPAddressValue src,dst, ByteValue
		  prot);
	public override bool Equals(PolicyCondition condition);
	public override bool Intersects(PolicyCondition condition);
	public override bool Contains(PolicyCondition condition);

	}
};
```

The IPAddress class is taken from System.Net namespace. It support both v4 and v6 IPAddresses. However, for one particular condition, all the condition fields have to be interpreted in the context of one address family. In other words, it will raise a runtime exception if the source address is a v4 address but the destination address is a v6 address.

Property	
Name	Layer
Description	The specific IP layer at which this condition is to be applied. The possible IP layers are as following: • **InboundIPPacket:** This layer is called after a just after the IP header has been parsed and just before any header processing takes place on received IP packet. IPSec decryption and reassembly will not have occurred at this point. • **OutboundIPPacket:** This layer is called just before a sent packet is evaluated for fragmentation. By the time this layer is called, all IP header processing is complete and all extension headers are in place. IPSec authentication and encryption will have already occurred at this time.

	• **InboundIPFragment:** This layer is called for every received fragment. Non-fragmented packets that are received will not be called out for this layer. • **OutboundIPFragment:** This layer is called for every sent and forwarded fragment. If a sent IP packet is not fragmented, it will not be called out for this layer. • **IPForward:** This layer is called for each forwarded packet.
Access	Read/Write

Property	
Name	SourceAddress
Description	This value is used to match the source address field in the IP header.
Access	Read/Write

Property	
Name	DestinationAddress
Description	This value is used to match the destination address field in the IP header.
Access	Read/Write

Property	
Name	Protocol
Description	This value is used to match the protocol field in the IP header.
Access	Read/Write

Property	
Name	PacketLength
Description	This value is used to match the packet length field in the IP header.
Access	Read/Write

Property	
Name	NetworkInterface
Description	Specify the network interface on which this condition will be matched. If the layer property is set to be IPForward, it will only match the receiving interface of the forwarded packets when the rule direction is set to Inbound and the outgoing interface when the rule direction is set to Outbound.

Access	Read/Write

System.Firewall.Policy.TransportCondition

```
namespace System.Firewall.Policy
{
    public abstract class TransportCondition : PolicyCondition
    {
        public enum TransportLayer {
            Inbound,
            Outbound
        };
            // The following are the conditions that are availabe at the
transport layer
            // via context.
            Public TransportLayer TransportLayer { get { } set { } }
            public IPAddressValue SourceAddress { get { } set { } }
            public IPAddressValue DestinationAddress { get { } set { } }
            public ByteValue Protocol { get { } set { } }
            // There may be more conditions to be exposed by the firewall
platform.
            protected TransportCondition();
            protected TransportCondition(TransportLayer layer,
IPAddressValue srcAddr, IPAddressValue dstAddr);
        }
    public class UDPCondition : TransportCondition
    {
        public UInt16Value SourcePort { get { } set { } }
        public UInt16Value DestinationPort { get { } set { } }
            public UDPCondition();
            public UDPCondition(TransportLayer layer, IPAddressValue
             srcAddr, IPAddressValue dstAddr, UInt16Value srcPort,
             UInt16Value dstPort);
```

```
        public override bool Equals(PolicyCondition val);
        public override bool Intersects(PolicyCondition val);
        public override bool Contains(PolicyCondition val);
    }
            public class TCPCondition : TransportCondition
            {
                    [flags]
                    public enum TCPFlags
                    {
                            FIN = 1,
                            SYN = 2,
                            RST = 4,
                            PSH = 8,
                            ACK = 16,
                            URG = 32
                    }
                    public UInt16Value SourcePort { get { } set { } }
                    public UInt16Value DestinationPort { get { } set { } }
                    public TCPFlags Flags { get { } set { } }
                    // There may be more conditions to be exposed by the firewall
platform.
                    public TCPCondition();
                    public TCPCondition(TransportLayer layer, IPAddressValue
                     srcAddr, IPAddressValue dstAddr, UInt16Value srcPort,
                     UInt16Value dstPort);
        public override bool Equals(PolicyCondition val);
        public override bool Intersects(PolicyCondition val);
        public override bool Contains(PolicyCondition val);
            }
            public class ICMPCondition : TransportCondition
            {
```

```
                    public ByteValue ICMPType { get { } set { } }
                    public ByteValue ICMPCode { get { } set { } }
                    // There may be more conditions to be exposed by the firewall
platform.
                    public ICMPCondition();
                    public ICMPCondition(TransportLayer layer, IPAddressValue
srcAddr, IPAddressValue dstAddr, ByteValue icmpType, icmpCode);
    public override bool Equals(PolicyCondition val);
    public override bool Intersects(PolicyCondition val);
    public override bool Contains(PolicyCondition val);
        }
    );
```

ICMP v6 defines ICMP type and code differently than ICMP v4. The address family of the source and destination address determines if an ICMPCondition will be interpreted as ICMP v4 or v6.

Property	
Name	SourcePort
Description	This value is used to match the source port field in the TCP/UDP header.
Access	Read/Write

Property	
Name	DestinationPort
Description	This value is used to match the destination port field in the TCP/UDP header.
Access	Read/Write

Property	
Name	Flags
Description	This value is used to match the corresponding bits in the TCP flags field: FIN, SYN, RST, PSH, ACK, URG
Access	Read/Write

Property	
Name	ICMPType
Description	This value is used to match the type field in the ICMP header.
Access	Read/Write

Property	
Name	ICMPCode
Description	This value is used to match the code field in the ICMP header.
Access	Read/Write

System.Firewall.P licy.IPSecAuthorizati nCondition

```
namespace System.Firewall.Policy
{
    public class IPSecAuthorizationCondition : PolicyCondition
    {
      public IPSecAuthorizationCondition(RemoteIdentity RemoteID);
      public RemoteIdentity RemoteID { get { } }
      public IPAddressValue LocalAddress { get { } set { } }
      public UInt16Value LocalPort { get { } set { } }
      public UInt16Value Protocol { get { } set { } }
      public IPAddressValue RemoteAddress { get { } set { } }
      public UInt16Value RemotePort { get { } set { } }
    }
}
```

IPSecAuthorizationCondition matches both inbound and outbound packet IPSec context. The inspection is assumed to take place right after IPSec authentication completes. If the associated action is Permit, then the IPSec SA will be established and traffic will be secured. If the action to take is Deny, no SA will be created and the IPSec main mode negotiation will fail.

System.Firewall.Policy.ApplicationCondition

```
namespace System.Firewall.Policy

{

    [flags]
    public enum NetworkAccessFlag
    {
        Client = 1,
        Server = 2,
        ClientAndServer = 3,
        Multicast = 4
    }

    public abstract class ApplicationCondition : PolicyCondition
    {
                    public ApplicationIDValue Application { get { } set { } }

                    public IPrincipalValue LocalUser { get { } set { } }

                    // There may be more conditions to be exposed by the firewall
platform.

                    public ApplicationCondition();

                    public ApplicationCondition(ApplicationIDValue app,
IPrincipalValue luser);
            }

            public class AuthorizationCondition : ApplicationCondition
            {

                    // The following conditions are matched against values passed
down through

                    // winsock calls like connect or listen.

                    public NetworkAccessFlag { get { } set { } }

                    public IPAddressValue LocalAddress { get { } set { } }

                    public IPAddressValue RemoteAddress { get { } set { } }

                    public ByteValue Protocol { get { } set { } }

                    public UInt16Value LocalPort { get { } set { } }

                    public UInt16Value RemotePort { get { } set { } }

                    public RemoteIdentityValue RemoteID { get { } set { } }

            }

            public enum PromiscuousMode

            {
```

```
                AllIP = 1,
                AllMulticast = 2,
                IGMPMulticast = 3
            }
            public enum ResourceType
            {
                UDPPort,
                TCPPort,
                Raw
            }
            public class ResourceAssignmentCondition : ApplicaitonCondition
            {
                public IPAddressValue LocalAddress { get { } set { } }
                // if protocol is not TCP/UDP, this is assumed to be a Raw
socket case.
                public ResourceType ResourceType { get { } set { } }
                public UInt16Value ResourceValue { get { } set { } }
                public PromiscuousMode PMode { get { } set { } }
            }
        }
```

ApplicationCondition matches the conditions that are exposed by the application layer enforcement. This is the main engine for providing application and user based firewall policies.

Property	
Name	Application
Description	This is to match packets that are generated / received by this application.

Access	Read/Write

Property	
Name	LocalUser
Description	This is to match packets that are generated / received by this user.
Access	Read/Write